Efficiently Solving Hybrid Logic/Optimization Problems

Through Generalized Conflict Learning

Hui Li and Brian Williams
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{huili, williams}@mit.edu

Abstract

An increasing range of problems in Artificial Intelligence and Computer Science, such as autonomous vehicle control and planning with resources, are formulated through a combination of logical, algebraic and cost constraints. Their solution requires a hybrid mixture of logical decision techniques and mathematical optimization. Using Disjunctive Programming (DP) to formulate these problems, we present a novel algorithm, called DP Conflict-Directed Branch and Bound (B&B), that efficiently solves DP problems through a powerful three-fold method. First, during the search process, *generalized conflict learning* learns qualitative descriptions (*conflicts)* for regions of the state space that are infeasible or sub-optimal. Second, *forward conflict-directed search* uses these qualitative descriptions to heuristically guide the forward step of the search, by moving away from regions of state space corresponding to known *conflicts*. Finally, *induced unit clause relaxation* automatically forms a strong relaxed problem from a subset of the unit clauses that are implied by the original problem. Our experiments on model-based temporal plan execution for cooperative vehicles demonstrate an order of magnitude speed-up over Mixed Integer Programming B&B.

1 Introduction

An increasing range of problems in Artificial Intelligence and Computer Science involve finding optimal solutions to problems that involve a rich combination of logical and algebraic constraints, and require a hybrid coupling of logical decision techniques with mathematical optimization to solve. Examples include planning with resources, autonomous vehicle control and verification of timed and hybrid systems. Focusing on the area of autonomous vehicle control, deep space explorers must choose between tasks and temporal orderings, while optimizing flight trajectories for fuel usage. On Earth, search and rescue units must construct and compare different vehicle trajectories around dangerous areas, such as a fire, on the approach to a trapped individual.

Each of these tasks involves designing an optimal state trajectory, based on a continuous dynamic model. At some point, they must satisfy additional logical constraints, such as mission tasks, task orderings and obstacle avoidance.

These Hybrid Logic/Optimization Problems (HLOPs) can be formulated in three ways: first, by introducing integer variables and corresponding constraints to Linear Programming (LP), known as Mixed Integer Programming (MIP) as in [Schouwenaars et al, 2001, Vossen et al, 1999, Kautz and Walser, 1999]; second, by augmenting LP with propositional variables so that the propositional variables can be used to "trigger" linear constraints, known as Mixed Logic Linear Programming (MLLP)[1] in [Hooker and Osorio, 1999] and LCNF[2] in [Wolfman and Weld, 1999]; third, by extending LP with disjunctions, without adding any variables or constraints, called Disjunctive Programming (DP) as in [Balas, 1979]. This paper builds upon the last option, DP, which combines the expressive power of propositional logic with that of LP, without the overhead of additional variables or constraints.

In this paper we introduce a novel algorithm for efficiently solving Disjunctive Programs called Conflict-Directed Branch & Bound (DP-CD-BB). It extends the Branch & Bound (B&B) algorithm by using logical inference to do relaxation, by abstracting the qualitative descriptions of the source of discovered infeasibility and sub-optimality as *conflicts* to guide the forward search, so as to prune the state space. Our experiments, comparing DP-CD-BB against Mixed Integer Programming B&B (MIP-BB), demonstrate a significant performance gain on model-based temporal plan execution for cooperative vehicles.

DP-CD-BB builds upon the Conflict-Directed Clausal LP Branch & Bound method in [Krishnan, 2004], which uses the same formulation as DP and learns infeasible states as conflicts to guide the search. The concept of conflict learning from sub-optimality draws from Activity Analysis in [Williams and Cagan, 1994], which reasons using qualitative abstractions of sub-optimal subspaces in order to guide the numerical methods away from subspaces with the same abstractions.

2 Problem Formulation

We use disjunctive programs to effectively capture both the continuous dynamics and control decisions present in

[1]MLLP is a generalization from MIP, but its main feature is the introduction of propositional variables.
[2]Optimization is not involved.

hybrid logic/optimization problems. Figure 1 depicts a simple example of an HLOP, introduced in [Schouwenaars et al, 2001]. Eq. (1) describes its DP formulation. In particular, this is an instance of a spatial reasoning problem, in which a vehicle has to go from point A to C, without hitting the obstacle B, while minimizing the fuel use.

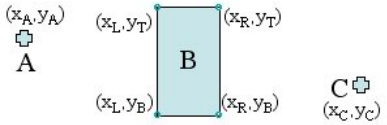


Figure 1. A simple example of an HLOP

$$\begin{aligned} &\text{Minimize} && f(\mathbf{x}) \\ &\text{Subject to} && \mathbf{g(x) \le 0} \qquad (1) \\ &\text{and} && x_i \le x_L \ \mathbf{V}\ x_i \ge x_R \ \mathbf{V}\ y_i \le y_B \ \mathbf{V}\ y_i \ge y_T\,, \\ & && \forall\ i = 1, \ldots, n \end{aligned}$$

In Eq. (1), **V** denotes logical *or*, **x** is a vector of decision variables that includes, at each time step i (=1,…,n), the position, velocity and acceleration of the vehicle; f(**x**) is a linear cost function in terms of fuel use; **g(x) ≤ 0** is a conjunction of linear constraints on vehicle dynamics, and the last constraint keeps the vehicle outside obstacle B at each time step i. In general, DP takes the form shown in Eq. (2):

$$\begin{aligned} &\text{Minimize} && f(\mathbf{x}) \\ &\text{Subject to} && \mathbf{\Lambda}_{i=1,\ldots,n} \left(\mathbf{V}_{j=1,\ldots,mi}\ C_j(\mathbf{x}) \le 0 \right) \qquad (2) \end{aligned}$$

where **x** is a vector of decision variables, f(**x**) is a linear cost function, and the constraints are a conjunction of n *clauses*, each of which (*clause* i) is a disjunction of m_i linear inequalities, $C_j(\mathbf{x}) \le 0$. DP reduces to a standard LP in the special case when $m_i=1$, $\forall i=1,\ldots,n$. In comparison with MIP, DP adds no overhead variables or constraints to represent logical decisions.

3 The DP-CD-BB Algorithm

The DP-CD-BB algorithm has four key features: First, Generalized Conflict Learning, which learns qualitative abstractions (*conflicts)* comprised of constraint sets that produce either infeasibility or sub-optimality; Second, Forward Conflict-Directed Search, which heuristically guides the *forward step* of the search away from regions of state space corresponding to known *conflicts*; Third, Induced Unit Clause Relaxation, which forms a relaxed problem from a subset of the unit clauses that are induced from the original problem; Fourth, Search Order: Best-first Search (BFS) versus Depth-first Search (DFS). In the following subsections, we develop these key features in detail, including examples and pseudo code.

DP-CD-BB builds upon B&B, which is frequently used by MIP, to solve problems involving both discrete and continuous variables. Instead of exploring the entire feasible set of a constrained problem, B&B uses bounds on the optimal cost, in order to avoid exploring subsets of the feasible set that it can prove are sub-optimal, that is, subsets whose optimal solution is not better than the *incumbent*, which is the best solution found so far. Pseudo code for B&B is shown in Figure 2.

GenericBB (original problem F)

```
1   incumbent U = +∞;
2   select a sub-problem Fi;
3   if (Fi is infeasible)
4      delete it;
5   else
6      compute the lower bound lb(Fi);
7   if (lb(Fi)≥ U)
8      delete Fi;
9   else if (the solution to Fi satisfies all the constraints of
    F)
10     U ← lb(Fi);
11  else
12     break Fi into sub-problems;
```

Figure 2. Pseudo code for generic Branch & Bound

The search tree of B&B for MIP branches by assigning values to the integer variables. In our case, B&B for DP branches by splitting clauses. At each node in the search tree, a relaxed LP is solved. *p'* is a relaxed LP of an optimization problem *p,* if the feasible region of *p'* contains the feasible region of *p*, and they have the same objective function. Therefore if *p'* is infeasible, then *p* is infeasible; if *p'* is solved with an optimal value *v*, the optimal value of *p* is guaranteed to be worse than *v*. B&B uses relaxed problems to obtain lower bounds of the original problem (assuming minimization).

3.1 Generalized Conflict Learning

Underlying the power of B&B is its ability to prune subsets of the search tree that correspond to relaxed sub-problems that B&B identifies as inconsistent or sub-optimal, as seen in line 4 and 8 in Figure 2.

In the related field of discrete constraint satisfaction, conflict-directed methods, such as dependency-directed backtracking [Stallman and Sussman, 1977], backjumping [Gaschnig, 1978], conflict-directed backjumping [Prosser, 1993] and dynamic backtracking [Ginsberg, 1993], dramatically improve the performance of backtrack (BT) search, by learning the source of each inconsistency discovered, and by using this generalization, called a *conflict*, to prune additional sub-trees that the conflict identifies as inconsistent.

To apply conflict learning to B&B, we note that B&B prunes subtrees corresponding to relaxed sub-problems that are sub-optimal and infeasible. Hence two opportunities exist for learning and generalized pruning. We exploit these opportunities by introducing the concept of *generalized conflict learning*, which extracts a qualitative

description from each pruned (fathomed) sub-problem that is infeasible or sub-optimal. This avoids exploring sub-problems with the same description in the future. Moreover, it is valuable to have the qualitative description as compact as possible, because the smaller the *conflict* is the larger the subspace to be pruned.

Each conflict can be of two types: (1) an irreducible set of constraints that is learned from infeasibility, or (2) an irreducible set of constraints that is learned from sub-optimality. A set of constraints is *irreducible* if removing any one of the constraints from the set resolves the infeasibility or sub-optimality. Note that there can be more than one irreducible sets (possibly with different cardinalities) involved in one infeasibility or sub-optimality, and a type-1 or type-2 conflict is not guaranteed to have the *minimal* cardinality. Hence the name *irreducible* instead of *minimal*. An *infeasibility conflict* (type 1) is an irreducible subset of the inconsistent constraints of an infeasible sub-problem. An example is the constraint set {a,c,d} in Figure 3(b). The sub-problem in Figure 3(a) is infeasible, but its constraint set is not an *infeasibility conflict*, as a proper subset of it, as in Figure 3(b), remains inconsistent. An *active* constraint of a feasible problem *S*, is a constraint that takes equality at *S*'s optimal solution x*. A *sub-optimality conflict* (type 2) is an irreducible subset of the active constraints of a feasible sub-problem whose optimal solution is not better than the *incumbent*. An example is the constraint set {c} in Figure 4(b). All the constraints are active in Figure 4(a), but the set {a,b,c,d} is not a *sub-optimality conflict*, as it can be reduced to Figure 4(b) without affecting the optimal solution x*.

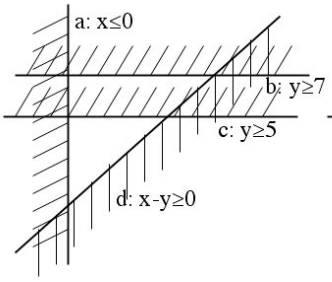
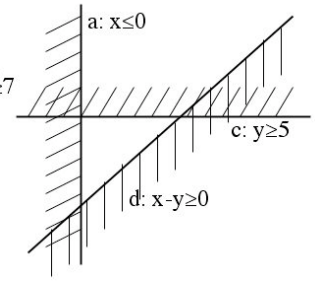


Figure 3(a). An infeasible sub-problem: constraint set {a,b,c,d} is not consistent.



(b) After removing constraint b, it is still infeasible.

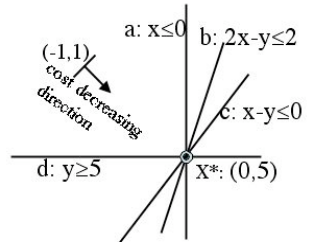
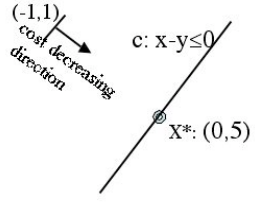


Figure 4(a).The optimal solution is X*. Constraints a, b, c and d are all active.



(b). After removing constraints a, b and d, X* stays the same.

To perform *generalized conflict learning* efficiently, the dual method of LP is used to extract a sub-problem's irreducible set. For infeasibility, this function is provided by the commercial software CPLEX: *getIIS()*, and its principle is explained in [Wolfman and Weld, 1999]. For sub-optimality, we introduce a novel approach based on the LP dual method. According to Complementary Slackness [Bertsimas and Tsitsiklis, 1997] from linear optimization theory or equivalently Kuhn-Tucker conditions for the linear case [Williams and Cagan, 1994], the non-zero terms of the optimal dual vector correspond to the irreducible set of active constraints at the optimal solution of the LP. Thus we use the dual optimal solution that is provided by the CPLEX function, *getDuals()*, to identify the *sub-optimality conflict.* The pseudo code is shown in Figure 5. After they are extracted, the conflicts are stored in a conflict database, *confDB,* with a timestamp that marks when they are discovered.

```
ExtractSubConf(LP problem p)
1.  solve p using CPLEX;
2.  if (p solved with an optimal solution) {
3.     dual ← getDuals();
4.     for (int  i=0; i<dual.length; i++)
5.        if (dual[i] !=0)
6.           subConf.add(constraint[i]); //constraint[i] is the
    corresponding constraint in p.
7.     return subConf;
8.  }else
9.     return null;
```

Figure 5. The function to extract sub-optimality conflicts

3.2 Forward Conflict-directed Search

The *forward conflict-directed search* heuristically guides the forward step of the search away from regions of the feasible space that are ruled out by known conflicts. Traditionally, conflicts are used in the backward step, such as dependency-directed backtracking [Stallman and Sussman, 1977], backjumping [Gaschnig, 1978], conflict-directed backjumping [Prosser, 1993], dynamic backtracking [Ginsberg, 1993] and LPSAT [Wolfman and Weld, 1999]. These backtrack search methods use conflicts to select backtrack points. In contrast, we use conflicts in forward search, to move away from known "bad" states. We generalize this idea to guiding B&B away from regions of state space that the known conflicts indicate are infeasible or sub-optimal. Our experiment results show that *forward conflict-directed search* significantly outperforms backtrack search on a range of cooperative vehicle plan execution problems.

The implementation , as seen in the pseudo code in Figure 6, includes three steps. 1. Conflict retrieval from *confDB*: only conflicts that are discovered after the creation time of the node to be expanded,

nodeToExp.timestamp, are retrieved, because conflicts discovered before are resolved by the creation of this node or its parents. Note that a node in the search tree represents a DP problem, which is a partially assigned problem from the original DP problem. 2. Negation: the types of the linear constraints in each conflict are reversed (e.g. ≤ becomes ≥) and the relation between the constraints in each conflict becomes logical *or*, so that a conflict is turned into a clause, called a *conflict clause*. Recall that a conflict represents the region where no feasible solution or only sub-optimal solutions exist. Therefore a *conflict clause* denotes the regions where an optimal solution can lie. 3. Clause addition: conflict clauses, *confClauses*, are added to the clause set of the node to be expanded, *nodeToExp.clauseSet*. In this way, *nodeToExp* is updated.

```
ForwardCDSearch(confDB, nodeToExp)
1.  if (confDB(nodeToExp.timestamp) != null) {
2.      currConfs ← confDB(nodeToExp.timestamp);
3.      for (int  i=0; i<currConfs.length; i++)
4.          for (int j=0; j<currConfs[i].length; j++)
5.              confClauses[i].add(¬ currConfs[i][j]);
6.      nodeToExp.clauseSet.add(conClauses);
7.  }
8.  GenericBB(nodeToExp);
```

Figure 6. Pseudo code for forward conflict-directed search

3.3 Induced Unit Clause Relaxation

Relaxation is an essential tool for quickly characterizing a problem when the original problem is hard to solve directly; it provides bounds on feasibility and the optimal value of a problem, which are commonly used to prune the search space. Previous research [Hooker, 2002] typically solves Disjunctive Programs by reformulating them as Mixed Integer Programs, in which binary integer variables are used to encode the disjunctive constraints. A relaxed problem for a MIP consists of the continuous relaxation of the integer constraints.

An alternative way of creating a relaxed LP is to operate on the DP encoding directly, by removing all non-unit clauses from the DP (a unit clause is one that contains a single constraint). Prior work argues for the reformulation of DP as MIP relaxation, with the rationale that it allows the solver to use continuous relaxation on the (binary) integer variables, in contrast to ignoring the non-unit clauses. However, this benefit is at the cost of adding integer variables and constraints, which can significantly increase the dimensionality of the search problem. This cost is not incurred by the DP relaxation.

Our approach starts with the direct DP relaxation, hence drawing from its strength in terms of a smaller state space. We overcome the weakness of standard DP relaxation (loss of non-unit clauses) by adding to the relaxation additional unit clauses that are logically entailed by the original DP. In the experiment section we compare our *induced unit clause relaxation* with the MIP relaxation and show a profound improvement on a range of cooperative vehicle plan execution problems.

The strongest relaxed problem is constructed when all entailed unit clauses are added to the relaxed problem; however, finding all of them is NP hard. A relaxation is valuable only to the extent that it saves computation time. Hence we choose the middle ground of finding all unit clauses that can be quickly induced. From propositional theories, unit clauses can be induced quickly through unit propagation; we generalize this approach to DP.

To implement *induced unit clause relaxation*, as seen in pseudo code in Figure 7 and the example in Figure 8, three steps are included. 1. Each unique constraint in the clause set of the DP problem *p* is assigned a unique propositional symbol. 2. (Incremental) Unit Propagation, as presented in ITMS [Nayak and Williams, 1997], is used to induce unit clauses. 3. The relaxed LP problem is formed with all the unit clauses induced from *p* and the objective function of *p*.

```
UnitClauseRelax(DP problem p)
1.  symbClauseSet ← p.clauseSet; //assigning a unique
    propositional symbol to each unique linear inequality.
2.  unitClauseSet ← UnitPropagation(symbClauseSet);
3.  relaxLP.constraintSet ← unitClauseSet.convert();
    //converting symbols to linear inequalities to form the
    relaxed LP
4.  relaxLP.objective ← p.objective;
5.  return relaxLP;
```

Figure 7. Pseudo code for induced unit clause relaxation

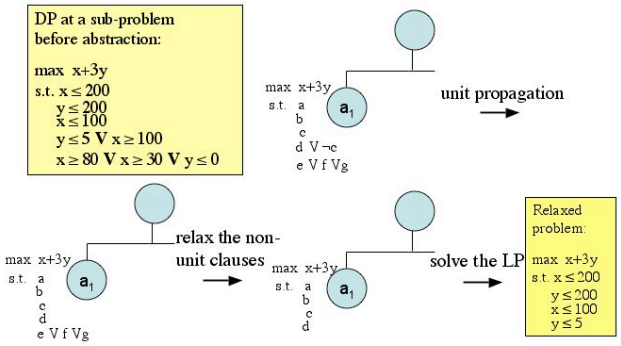


Figure 8. An example of induced unit clause relaxation

3.4. Search Order: Best-first versus Depth-first

For B&B it is known empirically that, in the average case, Best-first Search (BFS) performs better than Depth-first Search (DFS) in time efficiency. This is because BFS expands the search tree in the order of always exploring the most promising node, thus allows larger portions of the search tree to be pruned. However, BFS can take dramatically more memory space than DFS. Nevertheless, with conflict learning and forward conflict-directed search

the queue of the BFS search tree is significantly reduced. Our experimental results show that BFS can take memory space similar to DFS.

An additional issue for DP-CL-BB is that the concept of sub-optimality is rooted in maintaining an incumbent. Hence it can be applied to DFS but not to BFS (which does not have an incumbent). To evaluate these tradeoffs, our experiments in the next section compare the use of BFS and conflict learning from infeasibility only, with DFS and conflict learning from both infeasibility and from sub-optimality.

4 Experimental Performance Analysis

This section provides experimental results of the DP-CD-BB solver, compared with the benchmark MIP-BB, on a range of problems. We also compare the effect of several parameters, in particular, BFS versus DFS, infeasibility conflict learning versus sub-optimality conflict learning and forward search versus backtrack search. While each algorithmic variant terminates with the same optimal solution, a major result is that DP-CD-BB achieves an order of magnitude speed-up over MIP-BB. In addition, the difference in performance increases as the problem enlarges.

As the bulk of the computational effort expended by these algorithms is devoted to solving relaxed LP problems, the total number and average size of these LPs are representative of the total computational effort involved in solving the HLOPs. Note that extracting infeasibility conflicts and sub-optimality conflicts can be achieved as by-products of solving the LPs, and therefore does not incur any additional LP to be solved. We use the total number of relaxed LPs solved and the average LP size as our LP solver and hardware independent measures of computation time.

As explained in the first section and verified by our experiments, the MIP encoding enlarges the HLOP by adding overhead integer variables and constraints. Therefore, the average size of its LPs is larger than that of the LPs solved for the DP encoding. Experiments also show that the average sizes of relaxed LPs, solved by all the methods that use the DP encoding, are similar to each other. The data table is not listed due to space limit of the paper. To measure memory space use, maximum queue size is used.

We programmed MIP-BB, DP-CD-BB and its variations in Java. All used the commercial software CPLEX as the LP solver. Test problems were generated using a model-based temporal planner, performing multi-vehicle search and rescue missions. This planner takes as input a temporally flexible state plan, which specifies the goals of a mission, and a continuous model of vehicle dynamics, and encodes them in DP. The DP-CD-BB solver generates an optimal vehicle control sequence that achieves the constraints in the temporal plan. For each Clause/Variable set, 15 problems were generated and the average was recorded in the tables. These planning problems are tightly constrained and hence often "hard" problems, as studied in [Mitchell et al, 1992].

In Table 1, the number of relaxed LPs solved for each approach is recorded. The second row shows that MIP-BB solves more LPs than any other approach. The difference increases dramatically as the problem grows larger. The next three rows are dedicated to DP with BFS. The addition of infeasibility conflict learning (Inf) significantly outperforms without conflict learning (w.o. CL). The method using conflict-directed backtrack search (BT), which uses infeasibility conflicts to check consistency of a relaxed LP before solving it, performs dramatically worse than the method using forward conflict-directed search (Inf). The last five rows represent the variations of DP with DFS. Within these five rows, the method that solves the least relaxed LPs is the method with both infeasibility and sub-optimality conflict learning (Sub+Inf). The worst case is w.o. CL.

Consider BFS versus DFS, using only infeasibility conflict learning, BFS performs better than DFS, but the performance of DFS with Sub+Inf is close to BFS with Inf. For very large problems, DFS with Sub+Inf performs better. Under the same situation, either w.o. CL or with Inf or with BT, BFS solves less relaxed LPs than DFS, as explained before in section 3.4. As the only difference between DP with DFS without conflict learning and MIP-BB is in the formulation and relaxation methods, the significant improvement of the former over the latter verifies the statement in section 3.3.

For all tests, our DP-CD-BB algorithm, using either DP with BFS and infeasibility conflict learning, or DP with DFS and infeasibility plus sub-optimality conflict learning, performs the best and has a profound improvement over MIP-BB on large problems.

Clause / Variable		80 / 36	700/ 144	1492/ 300	2336/ 480
MIP-BB		31.5	2009	4890	8133
DP BFS	w.o. CL	24.3	735.6	1569	2651
	Inf	19.2	67.3	96.3	130.2
	BT	23.1	396.7	887.8	1406
DP DFS	w.o. CL	28.0	2014	3023	4662
	Inf	22.5	106.0	225.4	370.5
	BT	25.9	596.9	1260	1994
	Sub +Inf	22.1	76.4	84.4	102.9
	Sub	25.8	127.6	363.7	715.0

Table 1. Comparison on the number of relaxed LPs solved

In Table 2, all approaches have similar maximum queue sizes, except BFS w.o. CL and BFS with BT. As discussed in section 3.4, BFS generally takes more memory space than DFS, but when forward conflict-directed search is used, the search space is reduced and the corresponding queue is shortened. Note that although the methods using BT are conflict-directed, the queue size of the one with BFS is not largely reduced.

Clause / Variable		80 / 36	700/ 144	1492/ 300	2336/ 480
MIP-BB		8.4	30.8	46.2	58.7
DP BFS	w.o. CL	19.1	161.1	296.8	419.0
	Inf	6.4	18.3	38.4	52.5
	BT	15.6	101.7	205.1	327.8
DP DFS	w.o. CL	6.1	18.7	25.1	30.3
	Inf	6.5	21.4	45.0	57.3
	BT	6.1	18.4	23.5	28.1
	Sub +Inf	6.5	21.4	33.0	40.9
	Sub	6.5	21.6	38.7	47.0

Table 2. Comparison on the maximum queue size

5. Conclusion

Hybrid Logic/Optimization Problems can be encoded effectively using Disjunctive Programming (DP). This paper presented a novel algorithm, DP Conflict-Directed Branch and Bound, that efficiently solves DP problems through a powerful three-fold method, featuring *generalized conflict learning, forward conflict-directed search* and *induced unit clause relaxation*. The key feature of the approach is that infeasible or sub-optimal subsets of state space are reasoned using qualitative descriptions (*conflicts*), in order to heuristically guide the forward step of the search, by moving away from regions of state space corresponding to known *conflicts*. Our experiments on model-based temporal plan execution for cooperative vehicles demonstrated an order of magnitude speed-up over Mixed Integer Programming Branch and Bound.

References

[Balas, 1979] E. Balas. Disjunctive programming, *Annals of Discrete Mathematics* **5** 3-51.

[Bertsimas and Tsitsiklis, 1997] D.Bertsimas and J.N.Tsitsiklis. Introduction to Linear Optimization.

[Bitner and Reingold, 1975] J. Bitner and E. Reingold. Backtrack Programming Techniques, *Communications of the Association for Computing Machinery 18(11).*

[Gaschnig, 1978] J. Gaschnig Experimental Case Studies of Backtrack vs. Waltz-type vs. New Algorithms for Satisficing Assignment Problems. *The 2nd Canadian Conference on AI 268-277.*

[Ginsberg, 1993] M. Ginsberg, Dynamic backtracking, *Journal of Artificial Intelligence Research* ***1*** *25–46.*

[Hooker and Osorio, 1999] J.N.Hooker and M.A.Osorio. Mixed Logical/Linear Programming. *Discrete Applied Mathematics* ***96-97*** *395-442.*

[Hooker, 2002] J.N.Hooker, Logic, optimization and constraint programming, *INFORMS J. on Computing 14 295-321.*

[Krishnan, 2004] R. Krishnan. Solving Hybrid Decision-Control Problems Through Conflict-Directed Branch & Bound. *M.Eng. Thesis. MIT.*

[Mitchell et al, 1992] D. Mitchell, B. Selman, H. Levesque. Hard and easy distributions of SAT problems. *AAAI.*

[Nayak and Williams, 1997] P. Nayak, B. Williams. Fast Context Switching in Real-time Propositional Reasoning, *AAAI.*

[Prosser, 1993] P. Prosser. Hybrid algorithms for the constraint satisfaction problem, *Computational Intelligence 3, 268–299.*

[Ragno, 2002] R. Ragno. Solving Optimal Satisfiability Problems Through Clause-Directed A*. *M.Eng. Thesis. MIT.*

[Schouwenaars et al, 2001] T. Schouwenaars, B. De Moor, E. Feron, J. How. Mixed integer programming for multi-vehicle path planning. *European Control Conference.*

[Stallman and Sussman, 1977] R. Stallman and G. J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis, *Artificial Intelligence* ***9*** *135–196.*

[Vossen et al, 1999] T. Vossen, M. Ball, A. Lotem, D. Nau. On the use of integer programming models in AI planning. *IJCAI.*

[Williams and Cagan, 1994] B. Williams and J. Cagan. Activity Analysis: The Qualitative Analysis of Stationary Points for Optimal Reasoning. *AAAI.*

[Wolfman and Weld, 1999] S. Wolfman and D. Weld. The LPSAT engine & its application to resource planning. *IJCAI.*